A philosophical and technical comparison of Legion and Globus

A. S. Grimshaw
M. A. Humphrey
A. Natrajan

Grids are collections of interconnected resources harnessed to satisfy various needs of users. Legion and Globus are pioneering grid technologies. Several of the aims and goals of both projects are similar, yet their underlying architectures and philosophies differ substantially. The scope of both projects is the creation of worldwide grids; in that respect, they subsume several distributed systems technologies. However, Legion has been designed as a virtual operating system (OS) for distributed resources with OS-like support for current and expected future interactions among resources, whereas Globus has long been designed as a "sum of services" infrastructure, in which tools are developed independently in response to current needs of users. We compare and contrast Legion and Globus in terms of their underlying philosophy and the resulting architectures, and we discuss how these projects converge in the context of the new standards being formulated for grids.

1. Introduction

Grids are collections of interconnected resources harnessed to satisfy various user needs. The resources may be administered by different organizations and may be distributed, heterogeneous, and fault-prone. The manner in which users interact with these resources and the usage policies for the resources may vary widely. A grid infrastructure must manage this complexity so that users can interact with resources as easily and smoothly as possible.

Our definition, and indeed a popular one, is the following: *A grid system is a collection of distributed resources connected by a network.* A grid system (or, simply, a *grid*) gathers resources—whether they be desktop and hand-held hosts, devices with embedded processing resources (such as digital cameras and phones) or terascale supercomputers—and makes them accessible to users and applications. Access to these resources provides a means to reduce overhead and accelerate projects. A *grid application* can be defined as an application that operates in a grid environment or is on a grid system. Grid-system software (or *middleware*) is software that facilitates writing grid applications and manages the underlying grid infrastructure. The resources in a grid typically share at least some of the following characteristics:

- They are numerous.
- They are owned and managed by different organizations and individuals that may be mutually distrustful.
- They are potentially faulty.
- They have different security requirements and policies.
- They are heterogeneous; e.g., they have different CPU architectures, run different operating systems, and have different amounts of memory and disk storage.
- They are connected by heterogeneous, multi-level networks.
- They have different resource management policies.
- They are likely to be geographically separated (on a campus, in an enterprise, on a continent).

The above definitions of a grid and a grid infrastructure are necessarily general. What constitutes a "resource" is a deep question, and the actions performed by a user on a resource can vary widely. For example, a traditional

0018-8646/04/$5.00 © 2004 IBM

Report Documentation Page					*Form Approved* *OMB No. 0704-0188*
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE **MAR 2004**		2. REPORT TYPE			3. DATES COVERED **00-00-2004 to 00-00-2004**
4. TITLE AND SUBTITLE **A philosophical and technical comarison of Legion and Globus**					5a. CONTRACT NUMBER
					5b. GRANT NUMBER
					5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)					5d. PROJECT NUMBER
					5e. TASK NUMBER
					5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Virginia,Department of Computer Science,151 Engineer's Way,Cahrlottesville,VA,22094-4740**					8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)					10. SPONSOR/MONITOR'S ACRONYM(S)
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**					
13. SUPPLEMENTARY NOTES **The original document contains color images.**					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES **22**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

definition of a resource has been *machine*, or more specifically, *CPU cycles on a machine*. The actions users perform on such a resource can be running a job, checking availability in terms of load, and so on. These definitions and actions are legitimate, but limiting. Today, for example, resources can be as diverse as biotechnology applications, stock market databases, and wide-angle telescopes. The actions being run for each of these might be the following: *check whether license is available, join with user profiles,* and *procure data from specified sector*, respectively. A grid can encompass all such resources and user actions. Therefore, a grid infrastructure must be designed to accommodate these varieties of resources and actions without compromising basic principles such as ease of use, security, and autonomy.

A grid enables users to collaborate securely by sharing processing, applications, and data across systems with the above characteristics in order to facilitate collaboration, to speed up application execution, and provide easier access to data. More concretely, this means being able to

- Find and share data: Access to remote data should be as simple as access to local data. Incidental system boundaries should be invisible to users who have been granted legitimate access.
- Find and share applications: Many development, engineering, and research efforts consist of custom applications—permanent or experimental, new or legacy, public domain or proprietary—each with its own requirements. Users should be able to share applications with their own data sets.
- Find and share computing resources: Providers should be able to grant access to their computing cycles to users who need them without compromising the rest of the network.

In this paper, we compare two pioneering grid technologies: Legion and Globus. As members of the Legion project, we naturally have a deeper understanding of Legion than of Globus. However, we have attempted to present a careful, balanced, and symmetric comparison of the two projects from the literature we have found. To this end, we have avoided going into deep details of Legion in order to maintain parity with our understanding of Globus. In Section 2, we explain why we chose to compare Legion with Globus rather than other technologies. Under the broad definition of a grid given so far, we show that these two remain, at the time of this writing, as the only technologies that attempt to build a complete grid. In Section 3, we enumerate a set of requirements for grids and then describe how each project addresses each requirement, noting the relative importance of each requirement to each project. We also describe the design principles of each project. In that section and in the rest of this paper, our intention is to offer a constructive, informative differentiation to the community without criticizing the work of the Globus Toolkit**. Despite the differences between Legion and Globus, we respect the approach and successes of the Globus project and are currently working together on the community-defined Open Grid Services Architecture [(OGSA); see Section 6]. The specifics of each architecture are contained in Section 4. When referring to their philosophy and architecture, we refer to the two projects respectively as *Legion* and *Globus*, but if referring to their implementation, we refer to them respectively as *Legion 1.8* and *Globus 2.0*, the numbers denoting the latest versions available at the time of writing. Specifically, when referring to Globus, we discuss Version 2 of the toolkit (GT2), not Version 3 (GT3).[1] In Section 5, we present the current status of both projects, touching on commercial as well as academic deployments. In Section 6, we discuss both projects in the context of upcoming standards, specifically the OGSA being formulated by the Global Grid Forum (GGF**). We summarize this comparison in Section 7. We encourage readers to peruse the many Legion and Globus papers listed in the references and bibliography.

2. Background and related work

Over the years, there have been many definitions of what constitutes a grid. Below, we present a small list of the more visible definitions.

> *Users will be presented the illusion of a single, very powerful computer, rather than a collection of disparate machines. [. . .] Further, boundaries between computers will be invisible, as will the location of data and the failure of processors.*
>
> —Grimshaw [1]

> *We believe that the future of parallel computing lies in heterogeneous environments in which diverse networks and communication protocols interconnect PCs, workstations, small shared-memory machines, and large-scale parallel computers.*
>
> —Foster, Kesselman, and Tuecke [2]

> *A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.*
>
> —Foster and Kesselman [3]

[1] When this paper was written in early 2003, GT2 was the most prevalent version of the Globus architecture. Subsequently, the Globus Project released the new GT3 architecture. Although GT3 differs significantly from GT2 in architecture, its philosophical underpinnings are similar. Therefore, our comparison with GT2 continues to be highly relevant.

. . . a Grid is a system that: (i) coordinates resources that are not subject to centralized control . . . (ii) using standard, open, general-purpose protocols and interfaces . . . (iii) to deliver nontrivial qualities of service . . .

—Foster [4]

A Grid is a hardware and software infrastructure that provides dependable, consistent, and pervasive access to resources to enable sharing of computational resources, utility computing, autonomic computing, collaboration among virtual organizations, and distributed data processing, among others.

—Gentzsch [5]

. . . the overall grid vision [is] flexible access to compute, data, and unique network resources by providing flexible access and sharing of desktop PC resources.

—Chien [6]

A Grid system is a collection of distributed resources connected by a network. . . . Grid system software (or middleware) is software that facilitates writing Grid applications and manages the underlying Grid infrastructure.

—Grimshaw et al. [7]

With time, the definitions have become more and more general in order to encompass the wide capabilities we now expect from a grid. There is a clear tendency to define what constitutes a resource in a grid and what actions may be performed on those resources in more general terms. Moreover, several of the definitions present (and argue for or against) general characteristics of a grid. Evaluating currently available commercial and academic technologies against the characteristics mentioned in these and other definitions is a valuable survey exercise; however, we do not undertake it here. Instead, we compare Legion and Globus alone.

From the start, a common Legion and Globus goal was to build grids that would span administrative domains. This shared goal differentiates these projects from other distributed systems. For example, queuing systems, such as Network Queuing System (NQS) [8], Portable Batch System (PBS) [9], Load Sharing Facility (LSF) [10, 11], LoadLeveler* [12], Codine [13], and Sun Grid Engine (SGE) [13] were not initially designed to operate across administrative domains or even multiple clusters within a single domain. Although some of these queuing systems—LSF and SGE in particular—have adapted to multicluster configurations, their underlying design does not address grid goals. Likewise, systems such as Parallel Virtual Machine (PVM) [14], Message-Passing Interface (MPI) [15], Networks of Workstations (NOW) [16], Distributed Computing Environment (DCE) [17], and Mentat [18–20] were intended to enable writing parallel or distributed programs, but did not have any support for concerns such as wide-area security, multiple administrative domains, and fault tolerance. Some academic projects, such as Condor [21], Nimrod [22], and PUNCH [23] were designed in much the same manner as Legion and Globus. However, these projects did not encompass a diversity of resources and actions, as did Legion and Globus. For example, Condor provided support for location-independent running of a specific class of resources, namely applications; Nimrod provided fault-tolerance and monitoring of a specific class of resources, namely parameter-space applications; and PUNCH provided location-independent access to a specific class of resources, namely data files.

The Globe project [24] shares many common goals and attributes with Legion. For instance, both are middleware metasystems that run on top of existing host operating systems and networks, both support implementation flexibility, both have a single uniform object model and architecture, and both use class objects to abstract implementation details. However, the Globe object model is different. A Globe object is passive and is assumed to be potentially physically distributed over many resources in the system, whereas a Legion object is active and is expected to reside within a single address space. These conflicting views of objects lead to different mechanisms for inter-object communication. Globe loads part of the object (called a *local object*) into the address space of the caller, whereas Legion sends a message of a specified format from the caller to the callee. Another important difference is the notion of core object types. Legion core objects are designed to have interfaces that provide useful abstractions, enabling a wide variety of implementations. We are not aware of similar efforts in Globe. We believe that the design and development of the core object types define the architecture of Legion and ultimately determine its utility and success. Legion is designed to look like an operating system (OS) for grids, whereas Globe is designed to look more like an application environment. On the other hand, Globe differs much more from Globus because Globus does not have an underlying object model.

Although not intended for grid computing, the Common Object Request Broker Architecture (CORBA**) standard developed by the Object Management Group (OMG**) [25] shares a number of elements with the Legion architecture. Similar to the Legion idea of many possible object implementations that share a common interface, CORBA systems support the notion of describing the interfaces to active, distributed objects using an interface description language (IDL), and then linking the IDL to implementation code that might be written in any of a number of supported languages. Compiled object implementations rely on the services of an object request broker (ORB), analogous to the Legion run-time system, for performing remote method invocations. Despite these similarities, the different goals of the two systems result in different features. Whereas CORBA is more commonly used for business applications, such as providing remote database access from clients, Legion is intended for executing high-performance applications as well. This difference in vision manifests itself at all levels in the two systems—from the basic object model up to the high-level services provided. For example, where CORBA provides a simple RPC-based (remote procedure call) method execution model suited to client-server-style applications, Legion provides a coarse-grained dataflow method execution model, called a macro-dataflow [2] model, suitable for highly concurrent grid applications.

Finally, several commercial projects such as Entropia**, United Devices**, and Parabon** attempted and continue to attempt building large grids. However, unlike Legion and Globus, several of these projects impose restrictions such as the inability to run on a wide range of platforms, or code conversion of applications to one language or another. Such restrictions violate several of the grid principles to which both Legion and Globus adhere, as discussed in the next section.

3. Requirements and design principles

Legion and Globus share a common base of target environments, technical objectives, and target end users, as well as a number of similar design features. Both systems abstract access to processing resources, Legion via an interface called the *host object*, [3] and Globus via a service called the Globus Resource Allocation Manager (GRAM) interface [28]. Both systems also support applications developed using a range of programming models, including popular packages such as MPI, the standard (supported by most vendors) for interprocess communication on parallel computers. Despite these similarities, the systems differ significantly in their basic architectural techniques and design principles. Legion builds higher-level system functionality on top of a single unified object model [4] and set of abstractions to insulate the user/programmer from the underlying complexity of the grid. The Globus implementation is based on the combination of working components into a composite grid toolkit that fully exposes the grid to the programmer.

The Globus approach of adding value to existing high-performance computing services, enabling them to interoperate and work well in a wide-area distributed environment, has a number of advantages. For example, this approach takes advantage of code reuse and builds on user knowledge of familiar tools and work environments. However, a challenge associated with this sum-of-services approach is that, as the number of services grows in such a system, the lack of a common programming interface to Globus components and the lack of a unifying model of their interaction can have a negative impact on ease of use. Typically, end users must compensate for the system by providing their own mechanisms for service interoperability. By providing a common object programming model for all services, Legion enhances the ability of users and tool builders to use a grid computing environment effectively by employing the many services that are needed: schedulers, I/O services, applications, etc. Furthermore, by defining a common object model for all applications and services, Legion permits a more direct combination of services. For example, traditional system-level agents, such as schedulers, and normal application processes are both normal Legion objects exporting the standard object-mandatory interface. We believe in the long-term advantages of basing a grid computing system on a cohesive, comprehensive, and extensible design.

In this section, we contrast the Legion philosophy and architecture with our belief and understanding of the Globus philosophy and architecture. We start with the high-level design requirements for Legion and enumerate the design principles that guided its development. Following that, we describe the Globus philosophy and architecture. While we believe that the Legion and Globus teams largely agree on requirements, their differences lie in the emphasis or approach placed on each requirement. In this section, we therefore present the same requirements list as we did for Legion, but augmented with a discussion of each requirement in the context of Globus. We then enumerate the design principles that we believe guided the development of Globus. Whereas the requirements lists for Legion and Globus are identical, the design principles differ because of key differences in architectural approach. While we have attempted to differentiate the discussion of each requirement from its implementation (discussed in more detail in the architectural detail section), we have found it necessary at times to

[2] Macro-dataflow [19, 26, 27] is a data-driven computation model based on dataflow in which programs and subprograms are represented by directed graphs. Graph nodes are either method calls (procedure calls) or subgraphs. A node “fires” when data is available on all input arcs. In Legion, graphs are first-class and can be passed as parameters and manipulated directly [27]. Macro-dataflow lends itself to flexible communication between objects. For example, the model permits asynchronous calls, calls to methods where parameters may come in from yet other objects, and dispersal of results to multiple recipients.

[3] A Legion object runs in or is contained in a host object when it is executing. Thus, a host object is essentially a virtualization of what would now be called a hosting environment, as in Java** 2 Enterprise Edition (J2EE). The host object interface [29] includes methods for metadata collection, object instantiation, killing and suspending object execution, etc.

[4] The fact that Legion is object-based does not preclude the use of non-object-oriented languages or non-object-oriented implementations of objects. In fact, Legion supports objects written in traditional procedural languages such as C and Fortran [30] as well as object-oriented languages such as C++, Java, and Mentat Programming Language (MPL, a C++ dialect with extensions to support parallel and distributed computing) [20].

provide some implementation details to clarify the requirements and design discussion.

Legion requirements

Clearly, the minimum capability needed to develop grid applications is the ability to transmit bits from one machine to another—all else can be built from that. However, several challenges frequently confront a developer constructing grid applications. These challenges lead us to a number of requirements that any complete grid system must address. The designers of Legion believed, and continue to believe, that all of these requirements must be addressed by the grid infrastructure in order to reduce the burden on the application developer. If the system does not address these issues, they must be dealt with by the programmer, who is forced to spend valuable time on basic grid functions, thus needlessly increasing development time and costs. These requirements are high-level and independent of implementation; they are discussed in the following sections.

Security

Security covers a gamut of issues which include authentication, data integrity, authorization (access control), and auditing. If grids are to be accepted by corporate and government information technology (IT) departments, a wide range of security concerns must be addressed. Security mechanisms must be integral to applications and capable of supporting diverse policies. Furthermore, we believe that security must be firmly built in from the beginning. Trying to patch it in as an afterthought (as some systems are currently attempting to do) is a fundamentally flawed approach. We also believe that no single security policy is perfect for all users and organizations. Therefore, a grid system must have mechanisms that allow users and resource owners to select policies that fit particular security and performance needs while meeting local administrative requirements.

Global name space

The lack of a global name space for accessing data and resources is one of the most significant obstacles to wide-area distributed and parallel processing. The current multitude of disjoint name spaces greatly impedes the development of applications that span sites. All grid objects must be able to access (subject to security constraints) any other grid object *transparently* without regard to location or replication.

Fault tolerance

Failure in large-scale grid systems is and will be a fact of life. Machines, networks, disks, and applications frequently fail, restart, disappear, or otherwise behave unexpectedly. Forcing the programmer to predict and handle all of these failures significantly increases the difficulty of writing reliable applications. Fault-tolerant computing is known to be a very difficult problem. Nonetheless, it must be addressed, or businesses and researchers will not entrust their data to grid computing.

Accommodating heterogeneity

A grid system must support interoperability between heterogeneous hardware and software platforms. Ideally, a running application should be able to migrate from platform to platform if necessary. At a bare minimum, components running on different platforms must be able to communicate transparently.

Binary management and application provisioning

The underlying system should keep track of executables and libraries, knowing which ones are current, which ones are used with which persistent states, where they have been installed, and where upgrades should be installed. These tasks reduce the burden on the programmer.

Multilanguage support

Diverse languages will always be used, and legacy applications will need support.

Scalability

There are more than 400 million computers in the world today and more than 100 million network-attached devices (including computers). Scalability is clearly a critical necessity. Any architecture relying on centralized resources is doomed to failure. A successful grid architecture must adhere strictly to the distributed-systems principle: *The service demanded of any given component must be independent of the number of components in the system*. In other words, the service load on any given component must not increase as the number of components increases.

Persistence

Input/output (I/O) and the ability to read and write persistent data are critical for communicating between applications and for saving data. However, the current files/file-libraries paradigm should be supported, since it is familiar to programmers.

Extensibility

Grid systems must be flexible enough to satisfy current user demands and unanticipated future needs. Therefore, we feel that mechanism and policy must be realized by replaceable and extensible components, including (and especially) core system components. This model facilitates development of improved implementations that provide value-added services or site-specific policies while enabling the system to adapt over time to a changing hardware and user environment.

Site autonomy

Grid systems will be composed of resources owned by many organizations, each of which desires to retain control over its own resources. The owner of a resource must be able to limit or deny use by particular users, specify when it can be used, etc. Sites must also be able to choose or rewrite an implementation of each Legion component to best suit their needs. If a given site trusts the security mechanisms of a particular implementation, it should be able to use that implementation.

Complexity management

Finally, and significantly, complexity management is one of the biggest challenges in large-scale grid systems. In the absence of system support, the application programmer is faced with a confusing array of decisions. Complexity exists in multiple dimensions: heterogeneity in policies for resource usage and security, a range of different failure modes and different availability requirements, disjoint namespaces and identity spaces, and the sheer number of components. For example, professionals who are not IT experts should not have to remember the details of five or six different file systems and directory hierarchies (not to mention multiple user names and passwords) in order to access the files they use on a regular basis. Thus, providing the programmer and system administrator with clean abstractions is critical to reducing their cognitive burden.

Legion design principles

To address these basic grid requirements we developed the Legion architecture and implemented an instance of that architecture, the Legion run-time system [29, 31, 32]. The architecture and implementation were guided by the following design principles, which were applied at every level throughout the system.

Provide a single-system view

With today's operating systems, we can maintain the illusion that our local area network is a single computing resource. But once we move beyond the local network or cluster to a geographically dispersed group of sites, perhaps consisting of several different types of platforms, the illusion breaks down. Researchers, engineers, and product development specialists (most of whom do not want to be experts in computer technology) are forced to request access through appropriate gatekeepers, manage multiple passwords, remember multiple protocols for interaction, keep track of where everything is located, and be aware of specific platform-dependent limitations (this file is too big to copy or transfer to that system; that application runs only on a certain type of computer, etc.). Recreating the illusion of a single computing resource for heterogeneous, distributed resources reduces the complexity of the overall system and provides a single namespace.

Provide transparency as a means of hiding detail

Grid systems should support the traditional distributed system transparencies: access, location, heterogeneity, failure, migration, replication, scaling, concurrency, and behavior. For example, users and programmers should not have to know where an object is located in order to use it (access, location, and migration transparency), nor should they need to know that a component across the country failed; they want the system to recover automatically and complete the desired task (failure transparency). This behavior is the traditional way to mask details of the underlying system.

Provide flexible semantics

Our overall objective was a grid architecture suitable to as many users and purposes as possible. A rigid system design in which policies are limited, tradeoff decisions are preselected, or all semantics are predetermined and hard-coded would not achieve this goal. Indeed, if we dictated a single system-wide solution to almost any of the technical objectives outlined above, we would preclude large classes of potential users and uses. Therefore, Legion allows users and programmers as much flexibility as possible in the semantics of their applications, resisting the temptation to dictate solutions. Whenever possible, users can select both the kind and the level of functionality and choose their own tradeoffs between function and cost. This philosophy is manifested in the system architecture. The Legion object model specifies the functionality but not the implementation of the system core objects; the core system therefore consists of extensible, replaceable components. Legion provides default implementations of the core objects, although users are not obligated to use them. Instead, we encourage users to select or construct object implementations that answer their specific needs.

Reduce user effort

In general, there are four classes of grid users who are trying to accomplish some mission with the available resources: application end users, application developers, system administrators, and managers. We believe users want to focus on their jobs, e.g., their applications, and not on the underlying grid "plumbing" and infrastructure. Thus, for example, to run an application a user may type

```
legion_run my_application my_data
```

at the command shell. The grid should then take care of such details as finding an appropriate host on which to execute the application and moving around data and executables. Of course, the user may optionally be aware of and specify or override certain behaviors: for example, specify an operating system on which to run the job, name a specific machine or set of machines, or even replace the default scheduler.

Reduce "activation energy"

One of the typical problems in technology adoption is getting users to use it. If it is difficult to shift to a new technology, users will tend not to make the effort to try it unless their need is immediate and extremely compelling. This problem is not unique to grids—it is human nature. Therefore, one of our most important goals was to make grid technology easy to use; in chemistry terms, we kept the activation energy of adoption as low as possible. Thus, users can easily and readily realize the benefit of using grids and get the reaction going, creating a self-sustaining spread of grid use throughout the organization. This principle manifests itself in features such as "no recompilation" for applications to be ported to a grid and in support for mapping a grid to a local OS file system. Another variant of this concept is the motto "No play, no pay." The basic idea is that if a user does not need a feature, e.g., encrypted data streams, fault-resilient files, or strong access control, they should not have to pay the overhead of using it.

Do no harm

To protect their objects and resources, grid users and sites require grid software to run with the lowest possible privileges.

Do not change host operating systems

Organizations will not permit their machines to be used if their operating systems must be replaced. Our experience with Mentat [19] indicates, though, that building a grid on top of host operating systems is a viable approach. Furthermore, Legion must be able to run as a user-level process and not require root access. Overall, the application of these design principles at every level provides a unique, consistent, and extensible framework upon which to create grid applications.

Globus requirements

The philosophy underlying Globus can be found in the technical literature available on Globus and in several public pronouncements by key members of the design team. In this section, we describe the Globus philosophy by providing a structure similar to that of the previous section, which described the Legion philosophy. Again, we focus on GT2, not GT3, which is under development at the time of this writing. We first discuss Globus requirements, using the same list as for Legion, and this is followed by a discussion of Globus design principles. We show that there are many differences between Legion and Globus, both in the overall design principles and in the emphasis and approach to each of the requirements. The requirements are presented in the following sections.

Security

Security is an important facet of the Globus approach; both Legion and Globus have recognized and agreed that there will be diverse security mechanisms and policies in any grid. Globus addresses authentication and data integrity, but intentionally does not define an authorization model. Instead, Globus explicitly defers authorization to the underlying operating system.

Global name space

Naming is a key philosophical difference between Legion and Globus. The Globus approach is that a global name space is not a requirement for grids; rather, the combination of local naming mechanisms [e.g., UNIX** file system and UNIX process identifications (IDs)], uniform resource locators (URLs) (used to locate remote files), Internet Protocol (IP) addresses and the Domain Name System (DNS) (for naming remote resources), and Domain Names (DNs) (for humans) is sufficient. Location transparency is not a goal.

Fault tolerance

While both grid projects recognize that the system must be fault-tolerant, they differ in the degree to which the grid infrastructure itself masks the failures or errors. Globus focuses on low-level protocols for grid computing, arguing that the creation of robust, core low-level protocols enables other projects to create higher-level tools and protocols that will mask faults. Therefore, the Globus approach is not necessarily to implement fault tolerance, but rather to facilitate it.

Accommodating heterogeneity

Both projects agree that accommodating heterogeneity is a fundamental requirement.

Binary management and application provisioning

This requirement is viewed as a higher-level function and is therefore not directly part of the Globus Toolkit.

Multilanguage support

Both projects agree that multiple languages must be supported.

Scalability

While both projects recognize the need for scalability, the mechanisms and/or focus of scalability differ significantly. Since Legion provides more high-level services (a grid-enabled distributed file system, an integrated grid scheduler for all grid activities, etc.) than Globus, the Legion developers have had to deal directly with issues of scalability more often than the Globus developers. Since the Globus Toolkit focus has been "closer to the hardware," in many ways the scalability of Globus is a direct result of the scalability of the Internet. However, clearly both projects believe that scalability is an important issue.

Persistence

Persistence, much like scalability, is achieved in different ways in Legion and Globus. Generally, the approach in Legion is that persistence in grids requires grid-specific mechanisms, whereas persistence in Globus is largely achieved through non-grid-specific means, such as *inetd* (a Berkeley daemon program that listens for connection requests or messages for certain ports and starts server programs to perform the services associated with those ports). When a Globus user logs on to the grid, he realizes that he can access prior days' data because he knows where he left it and he knows that *gridftpd* is available there.

Extensibility

The toolkit approach explicitly allows for extensibility; as new requirements are identified, new toolkit components can be created. Arguably, however, individual users should not be able to choose a particular toolkit component and then reimplement or customize it for their individual requirements. In Legion, every functionality is intentionally extensible through the object-based design. In practice, this extensibility is achieved by means of operator overloading, inheritance, and (republishing) a new, open interface if the default implementation is not sufficient.

Site autonomy

Both projects are in close agreement on the need for strong site autonomy. In fact, both projects have devoted a significant amount of time arguing that *being part of a grid* does not mean that users can execute anything they want on any site. Rather, it is crucial that sites do not have to give up any rights to participate in a grid.

Complexity management

It is not clear to what extent Globus incorporates complexity management as a fundamental requirement of the grid infrastructure. For example, Globus users are expected to remember a lot more than Legion users (where their computations are currently executing, where their files are, etc.). In other words, if a Globus user cannot remember where certain files are, it is not clear how they might be found. In contrast, a Legion user is presented a location-independent, distributed file system abstraction, which is searchable. Although Legion users can find the physical locations of files if they wish to, we argue that the physical location is not actually what users do want. Abstracting the physical location of a resource is part of complexity management.

Globus design principles

Given that there are differences with regard to the focus and attainability of the requirements, it is not surprising that the guiding principles are different for the Legion project and the Globus project. From our interpretation, the philosophy of Globus is based on the following principles.

Provide a toolkit from which users can pick and choose

A set of architectural principles is not necessary, because it ultimately restricts the development of "independent solutions to independent problems." Similarly, having all components share some common mechanisms and protocols (perhaps above intellectual property) restricts individual development and the pick-and-choose deployment possibilities. By contrast, Legion developers believe that there is a core set of protocols that are fundamental to most grid services; as such, most new components in Legion should, to some degree, reuse the core protocols and functionality. This is perhaps the single biggest difference in philosophies between the two projects.

Focus on low-level functionality, thereby facilitating high-level tools (and general usability)

Globus is based on the principles of the "hourglass model." In the Globus architecture, the neck of the hourglass consists of the resource and connectivity protocols. Since the lower-level protocols are so critical to the success of the grid, the focus within the core Globus project itself is on these protocols. Other projects can then build higher-level services, such as a file replica manager and grid schedulers. In Legion, we believe that these higher-level functionalities are absolutely critical for the usability of the grid, so we provide default implementations (which can be replaced) of many of these higher-level functionalities.

Use standards whenever possible for both interfaces and implementations

The Globus developers have been very diligent in identifying candidate existing software or protocols that might be appropriate for grid computing, albeit with some necessary modifications. Key examples are a reliance on File Transfer Protocol (FTP) for data movement, Public Key Infrastructure (PKI) for authentication [and the Generic Security Services Application Programming

Interface (GSSAPI) as an API authentication], OpenSSL (an open implementation of the Secure Sockets Layer) for low-level cryptographic routines, and Lightweight Directory Access Protocol (LDAP) as the basis for information services. In contrast, Legion uses existing standards whenever possible; however, we have decided that certain standards, such as GSSAPI, are not necessarily appropriate, either because of limited endorsement outside the grid community, because their complexity was judged to be not worth the potential value, or because ultimately there was not a smooth fit (even with modifications) between the interface/protocol and the unique, heterogeneous, dynamic nature of grids.

Emphasize the identification and definition of protocols and services first, and APIs and software development kits next

The contribution of Globus, in some sense, is not the software, but rather the protocol: *A standards-based open architecture facilitates extensibility, interoperability, portability, and code sharing; standard protocols make it easy to define services that provide enhanced capabilities* [33]. Protocols are essential for interoperability. In contrast, the Legion emphasis has been, arguably, on the software itself. While we believe that the success of this approach is our ability to deliver a highly usable software product, historically we have not placed a strong emphasis on direct interoperability with other grid approaches. This emphasis is changing, particularly within Avaki** (for more on Avaki, see the section on Legion future directions).

Provide open-source community development

Recognizing the tremendous impact of the open-source movement, particularly Linux**, Globus has always strongly endorsed an open-source community development for the Globus Toolkit. Legion has been open-source for much of its development; however, we have generally found that in practice people would rather have deployable binary versions than source code itself.

Provide immediate usefulness

Legion required a sophisticated, working core functionality that would be utilized in many grid services. As such, it was very difficult to deliver only pieces of a grid solution to the community, so we could not provide immediate usefulness to the community without the entire product. In contrast, Globus recognized that certain short-term problems (such as single sign-on) could largely be solved by a small number of software artifacts. The result was that Globus provided immediate usefulness to the grid community. Other examples of this kind of immediate usefulness include the following:

- For computation, focusing on high-performance application requirements.
- For data, focusing on replicated, large datasets essentially accessible by FTP, downplaying the importance of nonlocal access to small files.
- Focusing on authentication instead of authorization.
- Treating *computation* differently from *data*, and promoting separate *computational grids* and *data grids*.

Do* not *provide a virtual machine abstraction

Apparently the virtual machine abstraction was considered in the early days of the Globus project. However, the virtual machine abstraction was viewed as an inappropriate model, . . . *inconsistent with our primary goals of broad deployment and interoperability*.[5] Additionally, the . . . *traditional transparencies are unobtainable* [33]. In contrast, in the Legion project, the virtual machine is precisely what is needed to mask complexity in the environment. This was a fundamental difference in the approach taken by Legion and the GT2 project.

Overall, while arguably the requirements are equivalent for Legion and Globus, the emphasis on these requirements within each project is quite different. More significantly, the principles of each project as discussed in this section clearly indicate the stark differences in how to best achieve the goals. In the next section, we provide more details on how each project attempts to satisfy the requirements of grid computing.

4. Architectural details

As is obvious from the previous sections, Legion and Globus overlap significantly in their goals. In fact, the overlap between these projects is far greater than the overlap between either of them and any of the queuing systems or parallel systems mentioned in the background section. As discussed above, the philosophical and architectural differences between Legion and Globus are significant. In this section, we further contrast those differences by providing details on the implementation of each system.

Legion architectural details

Legion started out with the *top-down* premise that a strong architecture is necessary to build an infrastructure of this scope. This architecture is based on communicating services, which are implemented as objects. An object is a stateful component in a container process. Much of the initial design time spent on Legion was to determine the underlying infrastructure and the set of core services over

[5] This principle of *not* providing a virtual machine abstraction has been a consistent theme in the design of Globus Toolkit 2 (GT2). In Globus Toolkit 3 (GT3), this theme has been discarded in favor of a container model that is essentially a virtual machine.

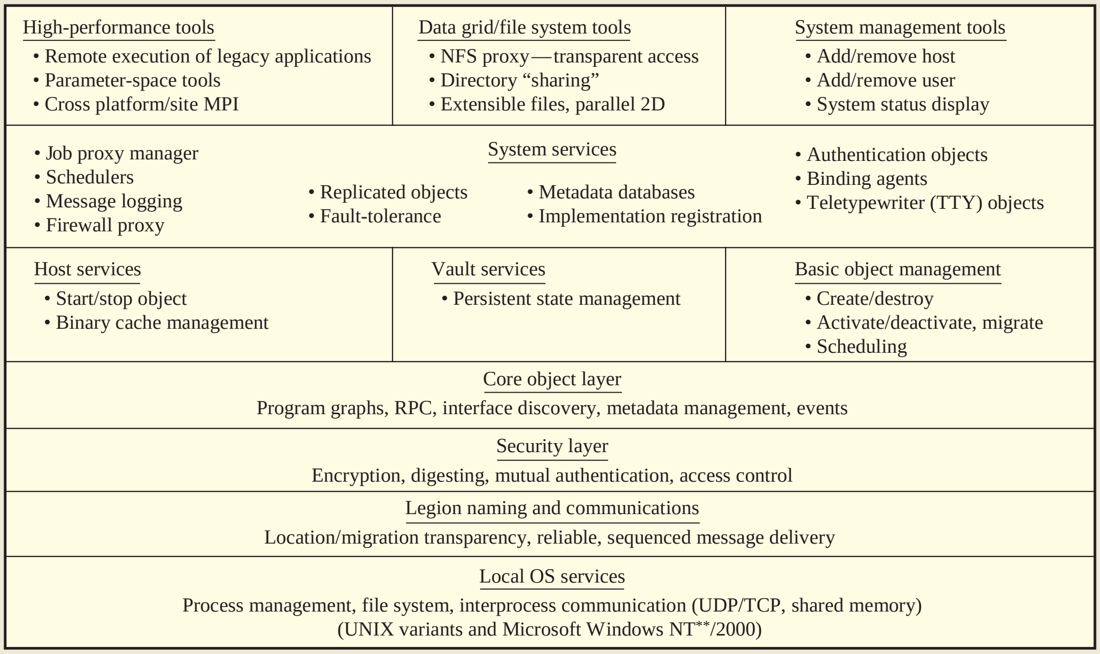

Figure 1

The Legion architecture viewed as a series of layers.

which grid services could be built [34–37]. Tasks involved in this design included the following:

- Designing and implementing a three-level naming, binding, and communication scheme. Naming is crucial in distributed systems and grids. The Legion scheme includes human-readable names such as attributes and directory-based pathnames to name objects (abstract names that do not include any location, implementation, or “number” information), and concrete object addresses.
- Building a remote method invocation framework based on dataflow graphs.
- Adding a security layer to every instance of communication between any services.
- Adding fault-tolerance layers.
- Adding layers for resource discovery through naming and adding mechanisms for caching name bindings (examples of bindings being IP addresses and ports for processes corresponding to services).

Legion designers have always believed that new services and tools must be built on top of a well-defined architecture. Much of the later development in Legion has been in terms of adding new kinds of services (two-dimensional files, specialized schedulers, firewall proxies, file export services, etc.) or tools that invoke methods of services (running an application [38], running parameter-space applications [39, 40], using a Web portal [41], etc.).

In **Figure 1**, we show a layered view of the Legion architecture. The bottom layer is the local operating system, or execution environment layer. This corresponds to true operating systems such as Linux, IBM AIX*, and Microsoft Windows NT/2000. The bottom layer, along with parts of the layers above, is also addressed by containers in hosting environments such as Sun Java** 2 Enterprise Edition (J2EE). We depend on process management services, file system support, and inter-process communication services delivered by the bottom layer, e.g., User Datagram Protocol (UDP), Transmission Control Protocol (TCP), or shared memory. Above the local operating services layer we build the Legion communications layer. This layer is responsible for object naming and binding as well as delivering sequenced arbitrarily long messages from one object to another. Delivery is accomplished regardless of the location of the two objects, object migration, or object failure. For example, object A can communicate with object B even while object B is migrating from Charlottesville to San Diego, or even if object B fails and subsequently restarts.

This is possible because of the Legion three-level naming and binding scheme, in particular the lower two levels.

The lower two levels consist of location-independent abstract names called Legion object identifiers (LOIDs) and object addresses specific to communication protocols, e.g., an IP address and a port number. The binding between a LOID and an object address can and does change over time. Indeed, it is possible for there to be no binding for a particular LOID at times when, for example, the object is not running currently. Maintaining the bindings at run time in a scalable way is one of the most important aspects of the Legion implementation [29].

Next is the security layer on the core object layers. The security layer implements the Legion security model [42, 43] for authentication, access control, and data integrity (e.g., mutual authentication and encryption on the wire). The core object layer [29, 31, 32] addresses method invocation, event processing (including exception and error propagation on a per-object basis [44]), interface discovery, and the management of metadata. Objects can have arbitrary metadata, such as the load on a host object or the parameters that were used to generate a particular data file.

Above the core object layer are the core services [29] that implement object instance management (*class managers*) and abstract processing resources (*hosts*), and storage resources (*vaults*). These are represented by base classes that can be extended to provide different or enhanced implementations. For example, the *host* class represents processing resources. It has methods to start an object given a LOID, a persistent storage address, and the LOID of an implementation to use, stop an object given a LOID, kill an object, and so on. There are derived classes for UNIX and Microsoft Windows** called *UNIXHost* and *NTHost* that respectively use UNIX processes and Windows spawn. There are also derived classes that interact with back-end queuing systems, *BatchQueueHost*, and that require the user to have a local account and run as that user, *PCDHost* [43, 45]. There are similar sets of derived types for vaults and class managers that implement policies (for example, replicated objects for fault tolerance or stateless objects for performance and fault tolerance [26, 44]) and interact with different resource classes.

Above these basic object management services are a whole collection of higher-level system service types and enhancements to the base service classes. These include classes for object replication for availability [44], message-logging classes for accounting or postmortem debugging, firewall proxy servers for securely transiting firewalls, enhanced schedulers [34], databases, called *collections*, that maintain information on the attributes associated with objects (these are used extensively in scheduling), job proxy managers that "wrap" legacy codes for remote execution [38, 40], and so on.

Finally, an application support layer contains user-centered tools for parallel and high-throughput computing, data access and sharing, and system management.

In the high-performance tool set there are tools to wrap legacy codes (`legion_register_program`) and execute them remotely (`legion_run`) both singly and in large sets (as in a parameter-space study [40]). Legion MPI tools support cross-platform, cross-site execution of MPI programs [39], and basic Fortran support [30] tools wrap Fortran programs for running on a grid.

The Legion integrated data grid support is focused on both extensibility and reducing the burden on the programmer [46–48]. In terms of extensibility, there is a basic file type (`BasicFile`) that supports the usual functions: `read, write, stat, seek`, etc. All other file types are derived from this type. Thus, no matter the file type, it can still be treated as a basic file and, for example, piped into tools that expect sequential files. There are two-dimensional files that support read/write operations on columns, rows, and rectangular patches of data (both primitive types as well as "structs"). There are file types to support unstructured sparse data, as well as parallel files in which the file has been broken up and decomposed across several different storage systems.

Data can be moved into the grid by either of two methods. It can be copied into the grid, in which case Legion manages the data and decides where to place it, how many copies to generate for higher availability, and where to place those copies. Alternatively, data can be exported into the grid. When a local directory structure is exported into the grid, it is mapped to a chosen path name in the global name space (directory structure). For example, a user can map data/sequences in his local UNIX file system into /home/grimshaw/sequences using the legion_export_dir command, `legion_export_dir data/sequences /home/grimshaw/sequences`. Subsequent access from anywhere in the grid (whether read or write) is done directly against the files in the user's UNIX file system (subject to access control, of course).

To simplify ease of use, the data grid can be accessed via a daemon that implements the Network File System (NFS) protocol. Therefore, the entire Legion namespace, including files, hosts, and so forth, can be mapped into local OS file systems. Thus, shell scripts, Perl scripts, and user applications can run unmodified on the Legion data grid. Further, the usual UNIX commands such as `ls` work, as does Microsoft Windows Exploring (which shows the directory structure and files.).

Finally, there are the user portal and system-management tools to add and remove users, add and remove hosts, and join two separate Legion grids together

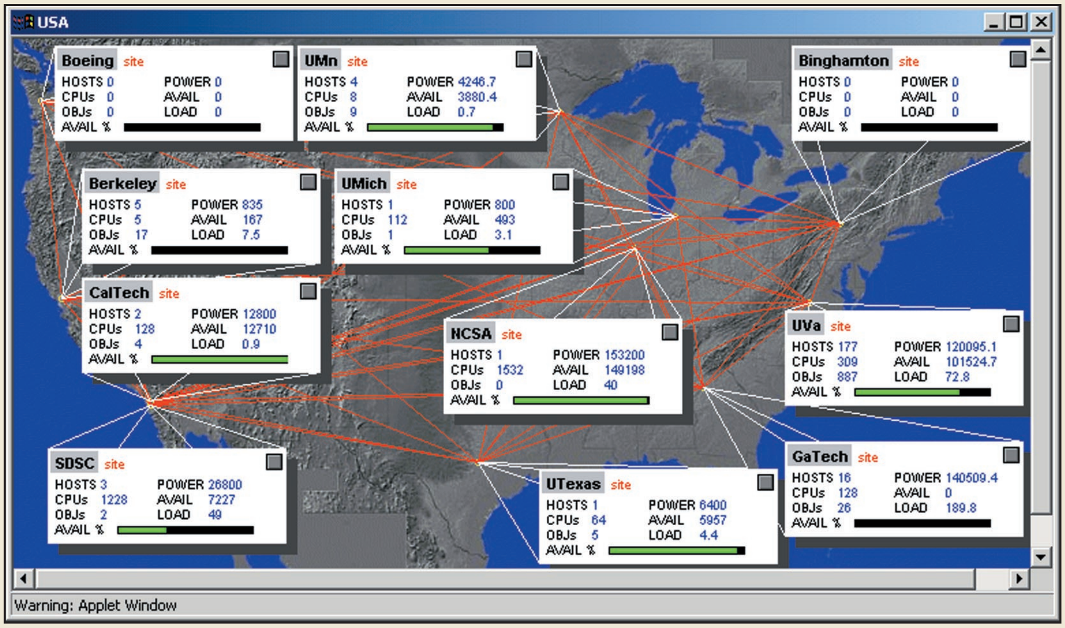

Figure 2

Legion system monitor running on NPACI-net in 2000 with the *USA* site selected. Clicking on a site opens a window for that site, with the individual resources listed. Sites can be composed hierarchically. Those resources, in turn, can be selected, and then individual objects are listed. *POWER* is a function of individual CPU clock rates and the number and type of CPUs. *AVAIL* is a function of CPU power and current load; it is what is available for use.

to create a grid of grids, etc. There is a Web-based portal interface for access to Legion [41] and a system status display tool that gathers information from system-wide metadata collections and makes it available via a browser (**Figure 2**).

The Web-based portal (**Figures 3–6**) allows an alternative, graphical interface to Legion. Using this interface (Figure 3), a user can submit an Amber job (a three-dimensional molecular modeling code) to NPACI-net (a University of Virginia Legion network) and not care at all where it executes. In Figure 4, we show the portal view of the intermediate output, where the user can copy files out of the running simulation and in which a three-dimensional molecular visualization plug-in called Chime is being used to display the intermediate results.

In Figure 5, we display the Legion job status tools. Using these tools, users can determine the status of all of the jobs they have started from the Legion portal and access their results as needed.

In Figure 6, we show the portal interface to the underlying Legion accounting system. We believed from very early on that grids must have strong accounting or they will be subject to the classic tragedy of the commons, in which everyone is willing to use grid resources, yet no one is willing to provide them. Legion keeps track of who used which resource (CPU, application, etc.), starting when, ending when, with what exit status, and with how much resource consumption. The data is loaded into a relational database management system, and various reports can be generated.

Globus architectural details

Globus started out with the *bottom-up* premise that a grid must be constructed as a set of tools developed from user requirements. This architecture is based on composing tools from a kit. Consequently, much of the initial design time was spent determining the user requirements for which grid tools could be built. Tasks involved in this design included building a resource manager to start jobs (assuming users had procured accounts beforehand on all of the machines on which they could possibly run), a tool and API for transferring files from one machine to another (used for binary and data transfer), tools for procuring credentials and certificates, and a service for collecting resource information about machines on a grid. The designers of Globus have always believed that new services and tools must be added to the existing set in such a way that users can combine any of the available

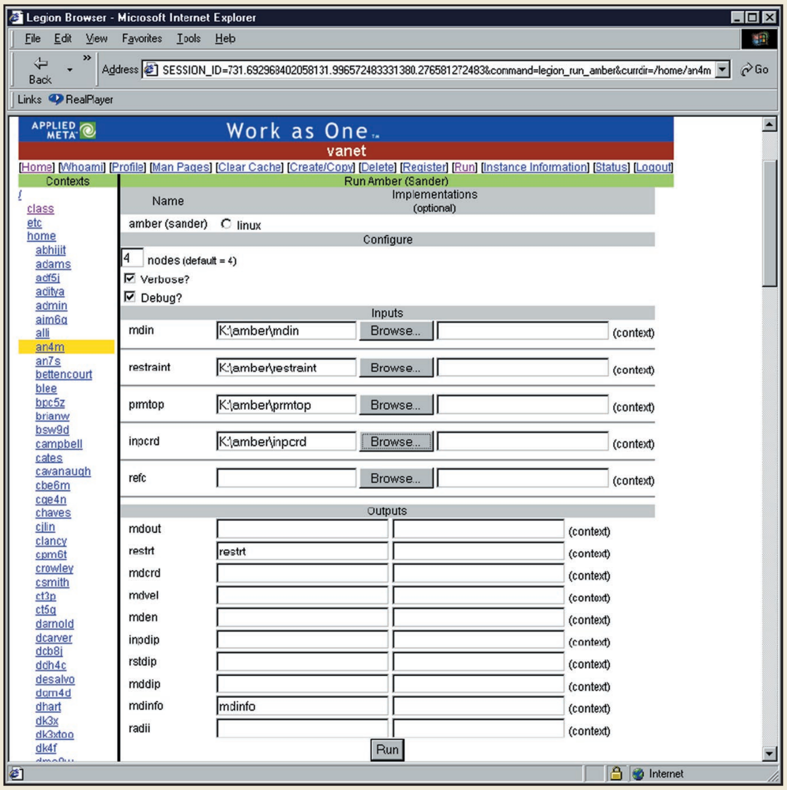

Figure 3

Job submission window for Amber using the Legion Web portal.

tools to get their work done. Much of the later development in Globus has been directed at composing these tools in order to achieve a specific goal.

In **Figure 7**, we show the early version (1997) of GT2, the Globus Toolkit, version 2 (adapted from Foster and Kesselman [49]). The *communications* module provides network-aware communications messaging capabilities. The implementation of the communications module in the Globus Toolkit was called *Nexus* [2]. The *resource location and allocation* module provides mechanisms for expressing application resource requirements for identifying resources that meet these requirements and for scheduling resources after they have been located. The *authentication* module provides a means by which to verify the identity of both humans and resources. In the Globus Toolkit, the GSSAPI was utilized in an attempt to make it unnecessary to know the actual underlying authentication technique, be it Kerberos (a centralized authentication system developed at MIT) or SSL (PKI). The *information service* module provides a uniform mechanism for obtaining real-time information about metasystem structure and status. The implementation of this module in the Globus Toolkit is called the *Metacomputing Directory Service* (MDS) [50], which builds upon the data representation and API of LDAP. The *data access* module is responsible for providing high-speed remote access to persistent storage, such as files. All of these modules lay on top of local OS services, as in Legion. Higher-level services utilize the components of the toolkit. Such higher-level services include parallel programming interfaces (e.g., MPICH/G2, an open implementation of the MPI standard developed at Argonne National Laboratory).

A more recent description of the Globus Toolkit, reflecting the evolution of the approach, is shown in **Figure 8** (adapted from Foster, Kesselman, and Tuecke [33]). At the bottom is the grid fabric layer, which provides the resources to which grid protocols mediate access. The Globus developers consider this to be generally lower than the components of the toolkit, with the exception of the Globus Architecture for Reservation and Allocation (GARA). The connectivity layer defines the core communication and authentication protocols required for grid-specific network transactions. Included in this layer from the Globus Toolkit is the Grid Security

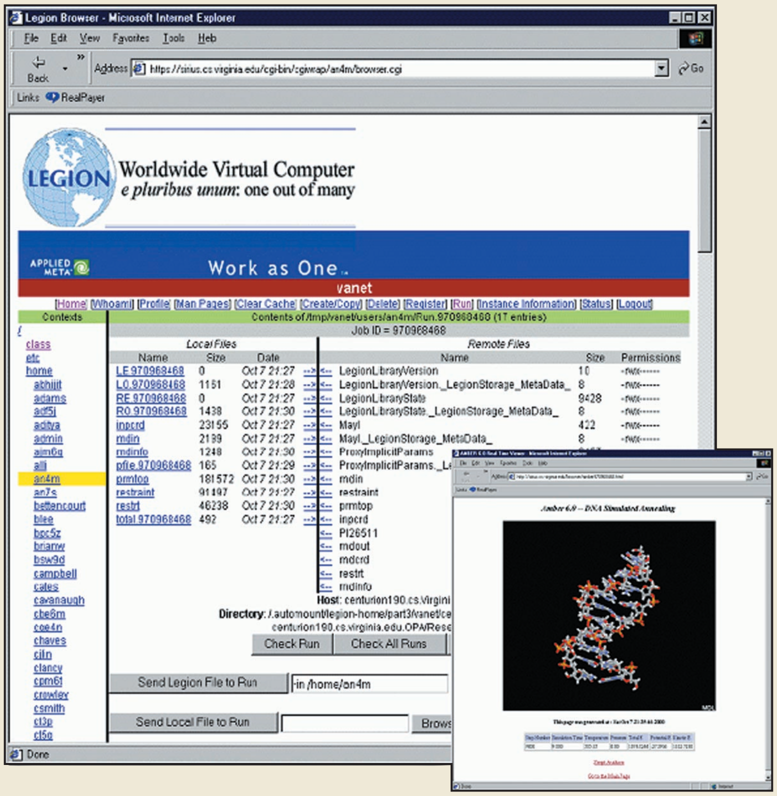

Figure 4

Chime plug-in displays updated molecule and application status.

Infrastructure (GSI) [51]. Above this is the resource layer, which defines protocols for secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. The Globus Toolkit functionality at this level includes a Grid Resource Information Protocol (GRIP), a resource information protocol; the Grid Resource Registration Protocol (GRRP), used to register resources with the Grid Index Information Servers; the Grid Resource Access and Management (GRAM) protocol, used to allocate and monitor resources; and GridFTP, which is used for data access. The collective layer is used to coordinate access to multiple resources, which, in terms of the Globus Toolkit, refers to MetaComputing Directory Service (MDS), supported by GRRP and GRIP. Finally, grid applications are at the very top.

Overall, the benefits and risks of either approach to software design—top-down or bottom-up—are well-known. With respect to grids, a bottom-up approach tends to result in early successes simply because the approach targets immediate user requirements. However, a risk with this approach is that the infrastructure may not be able to accommodate changing requirements. Another risk is that this approach may not scale as the number of tools or services increases, since an increasing number of pairwise protocols are necessary to ensure that the tools compose seamlessly. In contrast, the risk with a top-down approach is that initial successes are hard to come by, because at the beginning, designers focus on building an infrastructure with little or no end-user capability. Another risk is that the infrastructure being built could itself be so divergent from what users need that subsequent tools will not be useful. However, if the infrastructure is designed well, it tends to be flexible and amenable to a variety of tools and interfaces, all built over a common substrate. Moreover, the implementation of a new service or tool tends to be quick, since much of the complexity of the underlying substrate is abstracted away.

5. Future directions

Grid technology is becoming mature enough to move out of the indulgent environment of academia into the

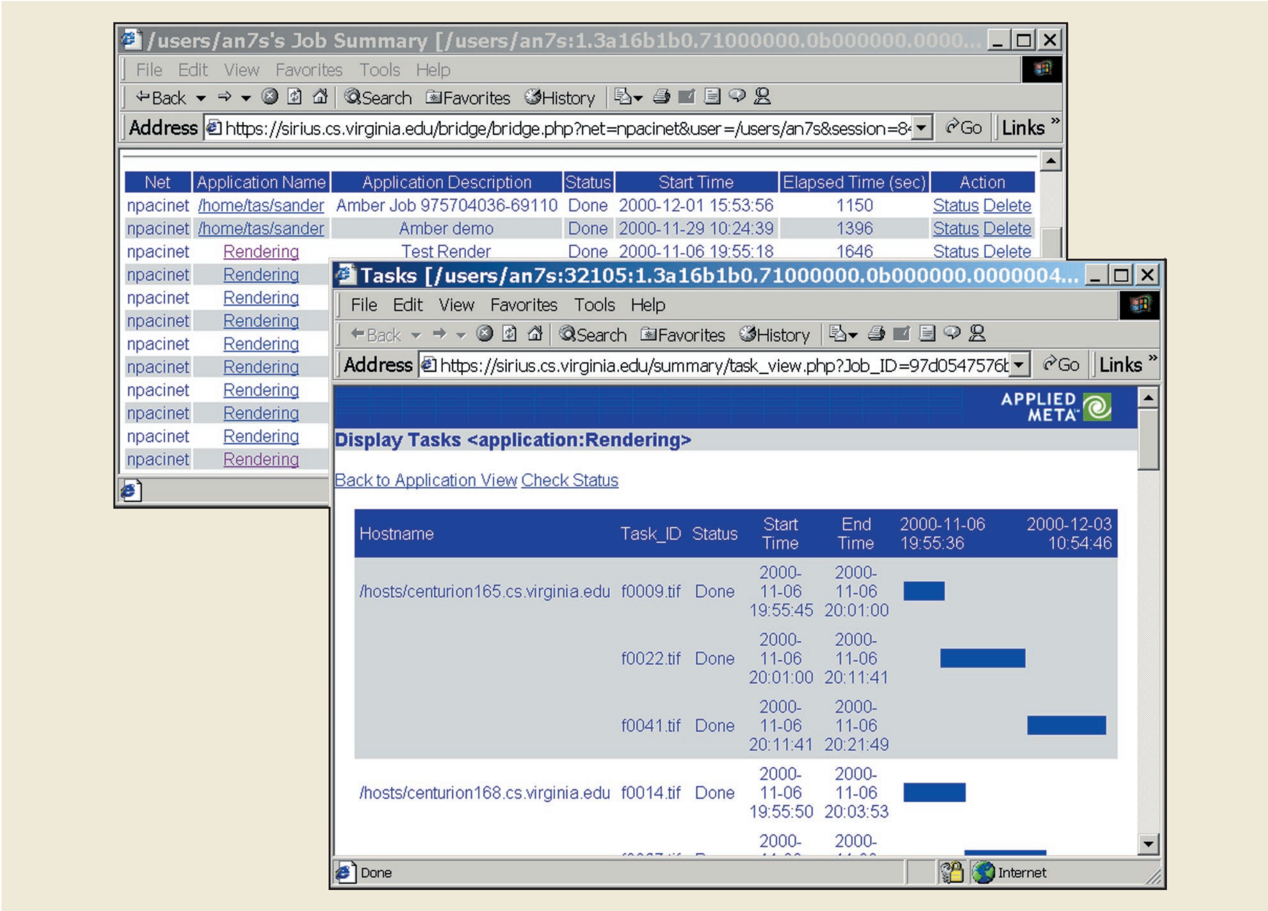

Figure 5

Legion job status tools accessible via the Web portal.

demanding world of commercial usage. In a commercial environment, nontechnological concerns such as standards acceptance, support personnel, open sources, and deployment model compete with the technological issues that we have discussed so far. Grids are large, infrastructure-style deployments. Therefore, while much of the technological discussion of the last decade or so has been valuable, it may now be time to focus on nontechnological issues as well. In this section, we present the initial approach of Legion and Globus to deployments and standards.

Legion future directions

From the outset of the Legion project, a technology transfer phase had been envisioned in which the technology would be moved from academia to industry. We felt strongly that grid software would move into mainstream business computing only with commercially supported software, help lines, customer support, services, and deployment teams. In 1999, Applied MetaComputing was founded to carry out the technology transition of Legion. In 2001, Applied MetaComputing raised $16 million in venture capital and changed its name to Avaki. The company acquired legal rights to Legion from the University of Virginia and changed its name to *Avaki*. Avaki was released commercially in September 2001. It is an extremely hardened, trimmed-down, focused-on-commercial-requirements version of Legion. While the name has changed, the core architecture and the principles on which it operates remain the same.

Many of the technological challenges faced by companies today can be viewed as variants of the requirements of grid infrastructures. The components of a project or product—data, applications, processing power, and users—may be in different locations from one another. Administrative controls set up by organizations to

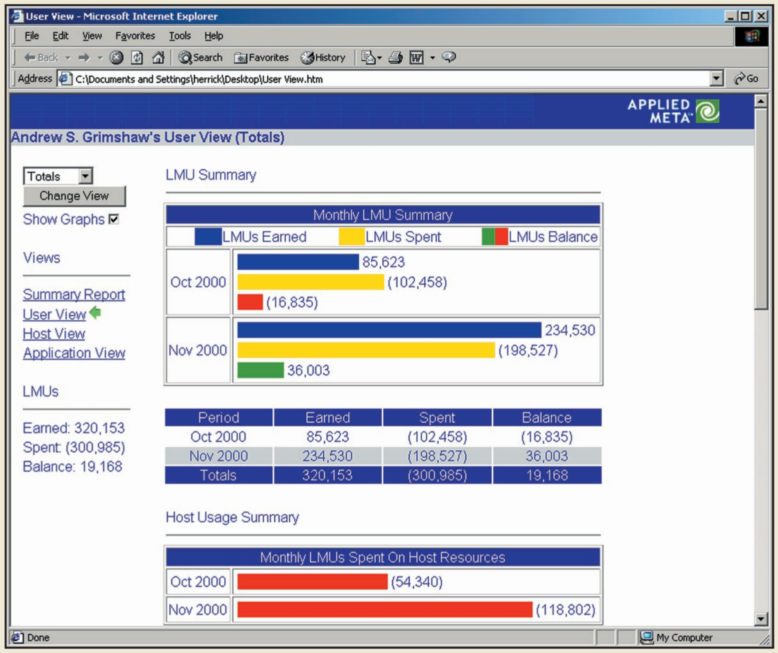

Figure 6

Legion accounting tool. Units are normalized CPU seconds. Displays can be organized by user, machine, site, or application. (An LMU is a Legion Monetary Unit; it is one CPU second normalized by the clock speed of the machine.)

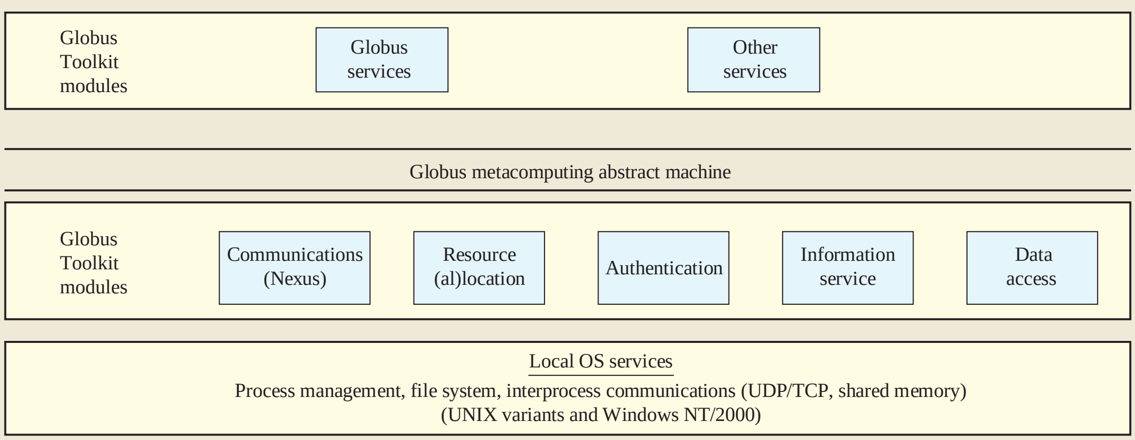

Figure 7

Globus Toolkit circa 1997 (adapted from Foster and Kesselman [49]).

prevent unauthorized accesses to resources hinder authorized accesses as well. Differences in platforms, operating systems, tools, mechanisms for running jobs, data organizations, and so on impose a heavy cognitive burden on users. Changes in resource usage policies and security policies affect the day-to-day actions of users. Finally, large distances act as barriers to the quick communication necessary for collaboration. Consequently, users spend too much time on the procedures for accessing a resource and too little time using the resource itself. These challenges lower productivity and hinder collaboration.

A successful technology is one that can smoothly make the transition from the comfort and confines of academia to the demanding commercial environment. Several academic projects are testament to the benefits of such transitions; often, the transition benefits not just the user community but the quality of the product as well. We believe that grid infrastructures are ready to make such a transition. Legion had been tested in nonindustry environments from 1997 onward, during which time we had the opportunity to rigorously test the basic model, scalability, security features, tools, and development environment. Further improvement required input from a more demanding community with a vested interest in using the technology for its own benefit. The decision to commercialize grids in the form of the Avaki 2.x product and beyond was inevitable.

Despite changes to the technology enforced by the push to commercialization, the basic technology in Avaki 2.x remains the same as Legion. All of the principles and architectural features discussed earlier continue to form the basis of the commercial product. As a result, the commercial product continues to meet the requirements outlined in the introduction. These requirements follow naturally from the challenges faced by commercial clients who attempt to access distributed, heterogeneous resources in a secure manner.

Beyond commercialization of the Legion technology in Avaki, the Legion team is focusing on issues of fault tolerance (autonomic computing) and security policy negotiation across organizations. This work is being done in the context of the open standards put forth by the Global Grid Forum (GGF): Open Grid Services Infrastructure (OGSI) and Open Grid Services Architecture (OGSA).

Globus future directions

As with Legion, the earlier days of Globus were largely characterized as being focused on interconnecting supercomputer centers across geographic boundaries more easily. As such, the community being courted by Globus was largely composed of U.S. Department of Energy sites and National Science Foundation sites (i.e., academic settings).

This situation began to change, at least by early 2001, when Globus began to seize upon opportunities outside the national labs and academia. On August 2, 2001, IBM made an announcement regarding their support of the U.K. E-science effort, in which the first open IBM support for the Globus project was announced. Later that year, at Supercomputing 2001, the Globus project announced the equivalent of "strategic partnerships" with 12 vendors (Compaq, Cray, SGI, Sun, Veridian, Fujitsu, Hitachi, NEC, Entropia, IBM, Microsoft, and Platform Computing).

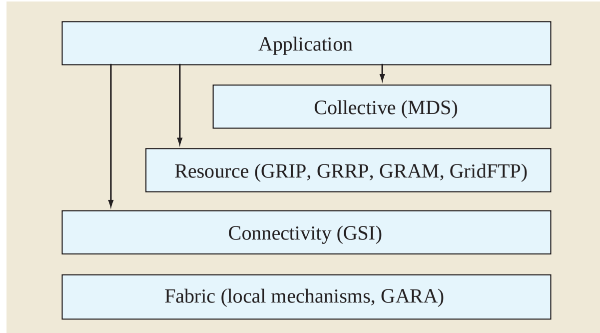

Figure 8

Globus Toolkit circa 2001, in the context of the Grid Protocol Architecture (adapted from Foster, Kesselman, and Tuecke [33]). (*MDS*: MetaComputing Directory Service; *GRIP*: Grid Resource Information Protocol; *GRRP*: Grid Resource Registration Protocol; *GRAM*: Grid Resource Access and Management; *GridFTP*: used for data access; *GSI*: Grid Security Infrastructure; and *GARA*: Globus Architecture for Reservation and Allocation.)

On April 12, 2002, Globus 2.0 was officially announced (a public beta version of Globus 2.0 was available as of November 15, 2001). Globus 2.0 is the basis for a number of grid activities, including the NSF Middleware Initiative (NMI), the E.U. DataGrid, and Grid Physics Network (GriPhyN) Virtual Data Toolkit (VDT).

A major development with regard to the future of the Globus Toolkit occurred on February 20, 2002, when the Globus Project and IBM announced their intention to create a new set of standards that would more closely integrate grid computing with Web services. This set of standards is the OGSA. This effort has since seen significant success and is now community-based and homed in the Global Grid Forum (see the next section for a more complete discussion). In January 2003, at GlobusWorld 2003, the Globus project announced the beta of the first version of the Globus Toolkit, GT3, to be OGSI-compliant.

6. Relationship to upcoming standards

OGSI Version 1.0 [52] and OGSA [53] are standards emerging from the GGF. These are the prevailing standards initiatives in grid computing and will, in our opinion, define the future of grid computing.

OGSI extends Web services via the definition of grid services. In OGSI, a grid service is a Web service that conforms to a particular set of conventions [54]. For example, grid services are defined in terms of standard Web Services Description Language (WSDL) [55–57] with some extensions, and exploit standard Web service binding technologies such as Simple Object Access Protocol (SOAP) [58–60] and Web Services Security (WS-Security). However, this set of conventions fundamentally sets grid services apart from Web services. Grid services introduce three fundamental differences:

- Grid services provide for named service instances and have a two-level naming scheme that facilitates traditional distributed systems transparencies.
- Services have a minimum set of capabilities, including discovery (reflection) services.
- There are explicitly stateful services with lifetime management.

OGSI has been developed in the OGSI working group in the GGF. The specification emerged from the standards process in the second quarter of 2003. It was submitted to the GGF editor as a recommendation track document and is now undergoing public comment. OGSI will be the basic interoperability layer (in terms of RPC, discovery, etc.) for a rich set of higher-level services and capabilities that are collectively known as the OGSA.

OGSA is being developed in the OGSA working group of the GGF, an umbrella working group within the GGF. The OGSA working group will spin off working groups to develop specialized service standards that, together, will realize a metaoperating system environment. The first of these working groups to form is the OGSA Security working group. Working groups on topics such as resources (hosts, storage, etc.), scheduling, replication, logging, management interfaces, and fault tolerance are anticipated.

The Legion authors enthusiastically support the OGSI effort. We support the OGSI/A standards efforts of the GGF for two primary reasons: the congruence of the OGSA with the Legion architecture and the importance of standards to users. The OGSA is highly congruent with both the existing Legion architecture and our architectural vision for the future. This congruence is not surprising given that both OGSA and Legion have the same objective: to create a metaoperating system for the grid. Our objective in building and designing Legion was

> *. . . to provide a solid, integrated, conceptual foundation on which to build applications that unleash the potential of so many diverse resources. The foundation must at least hide the underlying physical infrastructure from users and from the vast majority of programmers, support access, location, and fault transparency, enable inter-operability of components, support construction of larger integrated components using existing components, and provide a secure environment for both resource owners and users, and it must scale to millions of autonomous hosts* [1].

This vision is the mantra of OGSA as well. The means to the end are similar in both cases, and include the definition of base class services for basic building blocks of grids, hosts, storage, security, usage policies, failure detection mechanism, and failure recovery mechanisms and policies, e.g., replication services, and so on. The major architectural difference is in the RPC mechanism. OGSA and OGSI are based on Web services standards—SOAP/Extensible Markup Language (XML), WSDL, etc.—standards that did not exist when Legion was begun.

7. Summary

Legion and Globus are pioneering grid technologies. Both technologies share a common vision of the scope and utility of grids. To an outsider, i.e., a person interested in grids but not intimately familiar with either project, the differences between the two projects are unclear. Indeed, over the years, the authors of this paper have had to explain the differences several times in settings ranging from conference talks to informal dinner-table discussions. Doubtless, the architects of Globus have had similar experiences. This paper is an attempt at explaining those differences clearly.

The objective of this paper is not to disparage either of the two projects, but rather to contrast their architectural and philosophical differences, and thus educate the grid community about the choices available and the reasons why they were made at each step of the design. What is especially satisfying is the fact that the two projects are now converging toward a common, best-of-breed architecture that is certain to benefit designers and users of grid systems. Such a convergence would not have been possible without the rivalry of earlier days.

Acknowledgments

This work was partially supported by DARPA (Navy) Contract No. N66001-96-C-8527, DOE Grant DE-FG02-96ER25290, DOE Contract Sandia LD-9391, Logicon (for the DoD HPCMOD/PET program) DAHC 94-96-C-0008, DOE D459000-16-3C, DARPA (GA) SC H607305A, NSF-NGS EIA-9974968, NSF-NPACI ASC-96-10920, and a grant from NASA-IPG.

References

1. A. S. Grimshaw, "Enterprise-Wide Computing," *Science* **256**, 892–894 (August 1994).
2. I. Foster, C. Kesselman, and S. Tuecke, "The Nexus Task-Parallel Runtime System," *Proceedings of the First International Workshop on Parallel Processing*, Bangalore, India, 1994, pp. 457–462.
3. I. Foster and C. Kesselman, *Computational Grids, The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann Publishers, Inc., Los Altos, CA, 1999.
4. I. Foster, "What Is the Grid? A Three Point Checklist," *GridToday* **1**, No. 6 (July 2002); see *http://www.gridtoday.com/02/0722/020722.html*.
5. W. Gentzsch, "Response to Ian Foster's 'What Is the Grid?,'" *GridToday* **1**, No. 8 (August 5, 2002); see *http://www.gridtoday.com/02/0805/020805.html*.
6. A. Beck, Ed., "An Interview with Entropia's Andrew Chien," *GridToday* **1**, No. 5 (July 15, 2002); see *http://www.gridtoday.com/02/0715/100110.html*.
7. A. S. Grimshaw, A. Natrajan, M. A. Humphrey, M. J. Lewis, A. Nguyen-Tuong, J. F. Karpovich, M. M. Morgan, and A. J. Ferrari, "From Legion to Avaki: The Persistence of Vision," *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds., John Wiley & Sons, Hoboken, NJ, 2002; ISBN: 0-470-85319-0.
8. B. A. Kingsbury, "The Network Queueing System (NQS)," *Technical Report*, 1992, Sterling Software, 1121 San Antonio Road, Palo Alto, CA 94303; see *http://www.gnqs.org/oldgnqs/docs/papers/mnqs_papers/original_cosmic_nqs_paper.htm*.
9. A. Bayucan, R. L. Henderson, C. Lesiak, N. Mann, T. Proett, and D. Tweten, "Portable Batch System: External Reference Specification," *Technical Report*, November 1999, MRJ Technology Solutions, 10560 Arrowhead Dr., Fairfax, VA 22030; see *http://www.mrj.com/*.
10. S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: A Load Sharing System for Large, Heterogeneous Distributed Systems," *Software: Pract. & Exper.* **23**, No. 12, 1305–1336 (December 1993); *http://www.cs.ubc.ca/local/reading/proceedings/spe91-95/spe/spetoc.htm.*
11. S. Zhou, "LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems," presented at the Workshop on Cluster Computing, Tallahassee, FL, December 1992.
12. International Business Machines Corporation, "IBM LoadLeveler: User's Guide," IBM Corporation, September 1993.
13. F. Ferstl, "CODINE Technical Overview," Technical Report, April 1993, Genias Software, Dr. Gessler Strasse 20, D-93051 Regensburg, Germany; see *http://www.genias.de/*.
14. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine*, MIT Press, Cambridge, MA, 1994.
15. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1994.
16. T. Anderson, D. Culler, D. Patterson, and the NOW team, "A Case for NOW (Networks of Workstations)," *IEEE Micro* **15**, No. 1, 54–64 (February 1995).
17. H. Lockhart, Jr., *OSF DCE Guide to Developing Distributed Applications*, McGraw-Hill Inc., New York, 1994.
18. A. S. Grimshaw, "Easy to Use Object-Oriented Parallel Programming with Mentat," *IEEE Computer* **26**, No. 5, 39–51 (May 1993).
19. A. S. Grimshaw, J. B. Weissman, and W. T. Strayer, "Portable Run-Time Support for Dynamic Object-Oriented Parallel Processing," *ACM Trans. Computer Syst.* **14**, No. 2, 139–170 (May 1996).
20. A. S. Grimshaw, A. J. Ferrari, and E. West, "Mentat," *Parallel Programming Using C++*, G. V. Wilson and P. Lu, Eds., The MIT Press, Cambridge, MA, 1996, pp. 383–427.
21. M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor: A Hunter of Idle Workstations," *Proceedings of the Eighth International Conference on Distributed Computing Systems*, San Jose, CA, June 1988, pp. 104–111.
22. D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: A Tool for Performing Parameterized Simulations Using Distributed Workstations," *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing (HPDC-'95)*, Washington, August 1995, pp. 112–121.
23. N. H. Kapadia, R. J. Figueiredo, and J. A. B. Fortes, "PUNCH: Web Portal for Running Tools," *IEEE Micro* **20**, No. 3, 38–47 (May/June 2000).
24. M. van Steen, P. Homburg, and A. Tanenbaum, "The Architectural Design of Globe: A Wide-Area Distributed System," *Internal Report IR-422*, March 1997, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands; see *http://www.cs.vu.nl/~steen/globe/techreps.html*.
25. Object Management Group, "The Common Object Request Broker: Architecture and Specification," *Technical Report Rev. 2.0*, Object Management Group, Framingham, MA, July 1995 (updated July 1996).
26. A. Nguyen-Tuong, A. S. Grimshaw, and M. Hyett, "Exploiting Data-Flow for Fault-Tolerance in a Wide-Area Parallel System," *Proceedings of the 15th International Symposium on Reliable and Distributed Systems (SRDS-15)*, Ontario, Canada, October 1996, pp. 2–11.
27. C. L. Viles, M. J. Lewis, A. J. Ferrari, A. Nguyen-Tuong, and A. S. Grimshaw, "Enabling Flexibility in the Legion Run-Time Library," *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '97)*, Las Vegas, June 1997, pp. 265–274.
28. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, Orlando, FL, 1998, pp. 62–82.
29. M. J. Lewis, A. J. Ferrari, M. A. Humphrey, J. F. Karpovich, M. M. Morgan, A. Natrajan, A. Nguyen-Tuong, G. S. Wasson, and A. S. Grimshaw, "Support for Extensibility and Site Autonomy in the Legion Grid System Object Model," *J. Parallel & Distr. Computing* **63**, No. 5, 525–538 (May 2003).
30. A. J. Ferrari and A. S. Grimshaw, "Basic Fortran Support in Legion," *Technical Report CS-98-11*, March 1998, Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904; see *http://legion.virginia.edu/papers/CS-98-11.pdf*.
31. M. J. Lewis and A. S. Grimshaw, "The Core Legion Object Model," *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (HPDC-5)*, Syracuse, NY, August 1996, pp. 551–561.
32. A. Nguyen-Tuong, S. J. Chapin, A. S. Grimshaw, and C. Viles, "Using Reflection for Flexibility and Extensibility in a Metacomputing Environment," *Technical Report 98-33*, November 1998, Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904; see *http://legion.virginia.edu/papers/CS-98-33.rge.pdf*.
33. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int. J. Supercomputer Appl.* **15**, No. 3, 200–222 (2001).

34. S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw, "Resource Management in Legion," *Future Generation Computer Syst.* **15**, No. 5/6, 583–594 (October 1999).
35. A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds, Jr., "Legion: The Next Logical Step Toward a Nationwide Virtual Computer," *Technical Report CS-94-21*, June 1994, Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904; see *http://legion.virginia.edu/papers/CS-94-21.pdf*.
36. A. S. Grimshaw and W. A. Wulf, "Legion—A View from 50,000 Feet," *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, August 1996, pp. 89–99.
37. A. S. Grimshaw, J. B. Weissman, E. A. West, and E. Loyot, "Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems," *J. Parallel & Distr. Computing* **21**, No. 3, 257–270 (June 1994).
38. A. Natrajan, M. A. Humphrey, and A. S. Grimshaw, "Capacity and Capability Computing Using Legion," *Proceedings of the 2001 International Conference on Computational Science*, San Francisco, May 2001, pp. 273–283.
39. A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. A. Humphrey, A. J. Fox, A. S. Grimshaw, and C. L. Brooks III, "Studying Protein Folding on the Grid: Experiences Using CHARMM on NPACI Resources Under Legion," *Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC)*, San Francisco, August 2001, pp. 14–21.
40. A. Natrajan, M. A. Humphrey, and A. S. Grimshaw, "The Legion Support for Advanced Parameter-Space Studies on a Grid," *Future Generation Computer Syst.* **18**, No. 8, 1033–1052 (October 2002).
41. A. Natrajan, A. Nguyen-Tuong, M. A. Humphrey, M. Herrick, B. P. Clarke, and A. S. Grimshaw, "The Legion Grid Portal," *Grid Computing Environments, Concurrency and Computation: Pract. & Exper.* **14**, No. 13/15, 1365–1394 (2001).
42. S. J. Chapin, C. Wang, W. A. Wulf, F. C. Knabe, and A. S. Grimshaw, "A New Model of Security for Metasystems," *Future Generation Computer Syst.* **15**, No. 5/6, 713–722 (October 1999).
43. M. A. Humphrey, F. C. Knabe, A. J. Ferrari, and A. S. Grimshaw, "Accountability and Control of Process Creation in Metasystems," *Proceedings of the 2000 Network and Distributed Systems Security Conference (NDSS'00)*, San Diego, February 2000, pp. 209–220.
44. A. Nguyen-Tuong and A. S. Grimshaw, "Using Reflection for Incorporating Fault-Tolerance Techniques into Distributed Applications," *Parallel Processing Lett.* **9**, No. 2, 291–301 (1999).
45. A. J. Ferrari, F. C. Knabe, M. A. Humphrey, S. J. Chapin, and A. S. Grimshaw, "A Flexible Security System for Metacomputing Environments," *Proceedings of the Seventh International Conference on High-Performance Computing and Networking Europe (HPCN'99)*, Amsterdam, April 1999, pp. 370–380.
46. B. S. White, A. S. Grimshaw, and A. Nguyen-Tuong, "Grid-Based File Access: The Legion I/O Model," *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, Pittsburgh, August 2000, pp. 165–174.
47. B. S. White, M. P. Walker, M. A. Humphrey, and A. S. Grimshaw, "LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications," *Proceedings of Supercomputing 2001*, Denver, November 2001, p. 59.
48. J. F. Karpovich, A. S. Grimshaw, and J. C. French, "Extensible File Systems (ELFS): An Object-Oriented Approach to High Performance File I/O," *Proceedings of the Ninth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '94)*, Portland, OR, October 1994, pp. 191–204.
49. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int. J. Supercomputing Appl. & High Performance Computing* **11**, No. 2, 115–128 (1997).
50. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," *Proceedings of the Sixth IEEE Symposium on High-Performance Distributed Computing*, IEEE Computer Society Press, Washington, DC, 1997, pp. 365–375.
51. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, San Francisco, 1998, pp. 83–92.
52. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI) Version 1.0," Draft, *Global Grid Forum*, April 2003; see *http://www.gridforum.org/*.
53. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," *Open Grid Services Infrastructure WG*, Global Grid Forum, June 22, 2002; see *http://www.gridforum.org/*. Also appears as "Grid Services for Distributed System Integration," *IEEE Computer* **35,** No. 6, 37–46 (June 2002).
54. A. S. Grimshaw and S. Tuecke, "Grid Services Extend Web Services," *Web Services J.* **3**, No. 8 (August 2003); see *http://www.findarticles.com/cf_dls/m0MLV/8_3/106174061/p1/article.jhtml*.
55. R. Chinnici, M. Gudgin, J.-J. Moreau, and S. Weerawarana, "Web Services Description Language (WSDL), Version 1.2 Part 1: Core Language," *Technical Report*, W3 Consortium, June 2003; see *http://www.w3.org/TR/2002/WD-wsdl12-20020709/*.
56. M. Gudgin, A. Lewis, and J. Schlimmer, "Web Services Description Language (WSDL), Version 1.2 Part 2: Message Patterns," *Technical Report*, W3 Consortium, June 2003; see *http://www.w3.org/TR/2003/WD-wsdl12-patterns-20030611/*.
57. J.-J. Moreau and J. Schlimmer, "Web Services Description Language (WSDL), Version 1.2 Part 3: Bindings," *Technical Report*, W3 Consortium, June 2003; see *http://www.w3.org/TR/wsdl12-bindings/*.
58. N. Mitra, "SOAP Version 1.2 Part 0: Primer," *Technical Report*, W3 Consortium, June 2003; see *http://www.w3.org/TR/soap12-part0/*.
59. M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," *Technical Report*, W3 Consortium, June 2003; see *http://www.w3.org/TR/SOAP/*.
60. J.-J. Moreau, H. F. Nielsen, M. Gudgin, M. Hadley, and N. Mendelsohn, "SOAP Version 1.2 Part 0: Adjuncts," *Technical Report*, W3 Consortium, June 2003; see *http://www.w3.org/TR/soap12-part2/*.

Bibliography

Legion

A. S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Commun. ACM* **40,** No. 1, 39–45 (January 1997).

A. S. Grimshaw, A. J. Ferrari, G. Lindahl, and K. Holcomb, "Metasystems," *Commun. ACM* **41,** No. 11, 46–55 (November 1998).

A. S. Grimshaw, A. J. Ferrari, F. C. Knabe, and M. A. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale," *IEEE Computer* **32,** No. 5, 29–37 (May 1999).

L. J. Jin and A. S. Grimshaw, "From MetaComputing to Metabusiness Processing," *Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2000),* Saxony, Germany, December 2000, pp. 99–108.

A. Natrajan, M. A. Humphrey, and A. S. Grimshaw, "Grids: Harnessing Geographically-Separated Resources in a Multi-Organizational Context," *Proceedings of the 15th Annual Symposium on High Performance Computing Systems and Applications (HPCS 2001),* Windsor, Ontario, June 2001, pp. 25–32.

G. Stoker, B. S. White, E. L. Stackpole, T. J. Highley, and M. A. Humphrey, "Toward Realizable Restricted Delegation in Computational Grids," presented at the European High Performance Computing and Networking (HPCN 2001) conference, Amsterdam, June 2001.

W. A. Wulf, C. Wang, and D. Kienzle, "A New Model of Security for Distributed Systems," *Technical Report 95-34,* August 1995, Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904; see *http://www.cs.virginia.edu/dependability/bibliography/p34-wulf.pdf*.

Globus

G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf, "The Cactus Worm: Experiments with Dynamic Resource Selection and Allocation in a Grid Environment," *Int. J. High-Performance Computing Appl.* **15,** No. 4, 345–358 (2001).

J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," *Sixth Workshop on I/O in Parallel and Distributed Systems (IOPADS'99),* Atlanta, May 5, 1999.

R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, "A National-Scale Authentication Infrastructure," *IEEE Computer* **33,** No. 12, 60–66 (December 2000).

A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *J. Network & Computer Appl.* **23,** No. 3, 187–200 (2001).

K. Czajkowski, I. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids," *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8),* August 1999, pp. 219–228.

K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10),* San Francisco, August 2001.

I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke, "Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment," *Proceedings of the Fifth IEEE Symposium on High-Performance Distributed Computing,* IEEE Computer Society Press, Washington, DC, 1997, pp. 562–571.

I. Foster, J. Geisler, C. Kesselman, and S. Tuecke, "Managing Multiple Communication Methods in High-Performance Networked Computing Systems," *J. Parallel & Distr. Computing* **40,** No. 1, 35–48 (1997).

I. Foster, N. Karonis, C. Kesselman, G. Koenig, and S. Tuecke, "A Secure Communications Infrastructure for High-Performance Distributed Computing," *Proceedings of the Sixth IEEE Symposium on High-Performance Distributed Computing,* IEEE Computer Society Press, Washington, DC, 1998, pp. 125–136.

I. Foster and N. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems," *Proceedings of the 1998 Supercomputing Conference,* IEEE Computer Society Press, Washington, DC, 1998, pp. 1–11.

J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing* **5,** No. 3, 237–246 (2002).

K. Keahey, T. Fredian, Q. Peng, D. P. Schissel, M. Thompson, I. Foster, M. Greenwald, and D. McCune, "Computational Grids in Action: The National Fusion Collaboratory," *Future Generation Computer Syst.* **18,** No. 8, 1005–1015 (October 2002).

K. Keahey and V. Welch, "Fine-Grain Authorization for Resource Management in the Grid Environment," *Proceedings of Grid Computing (Grid2002): Third International Workshop,* Baltimore, MD, November 2002, pp. 199–206; see *http://www.globus.org/research/papers/gauth02.pdf.*

C. Lee, R. Wolski, I. Foster, C. Kesselman, and J. Stepanek, "A Network Performance Tool for Grid Computations," Proceedings of the Conference on Supercomputing, Portland, OR, 1999, pp. 260–267.

H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney, "File and Object Replication in Data Grids," *J. Cluster Computing* **5,** No. 3, 305–314 (2002).

Other

L. Smarr and C. E. Catlett, "Metacomputing," *Commun. ACM* **35,** No. 6, 44–52 (June 1992).

Received February 5, 2003; accepted for publication October 15, 2003

Andrew S. Grimshaw *Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, Virginia 22904 (grimshaw@cs.virginia.edu).* Dr. Grimshaw received his Ph.D. degree from the University of Illinois in 1988. He then joined the University of Virginia as an assistant professor of computer science and became an associate professor in 1994. He is the chief designer and architect of Mentat and Legion. Mentat is an object-oriented parallel processing system designed to simplify the task of writing parallel programs. Legion is a new collaborative project to realize the potential of the national information infrastructure (NII) by constructing a very large virtual computer that spans the nation. Legion addresses issues such as parallelism, fault tolerance, security, autonomy, heterogeneity, resource management, and access transparency in a multi-language environment. Dr. Grimshaw's research projects also include ELFS (extensible file systems), which addresses the I/O crisis brought on by parallel computers. He is the author or co-author of more than 50 publications and book chapters. In 2001, Dr. Grimshaw founded Avaki Corporation, which has commercialized grid technology.

Marty A. Humphrey *Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, Virginia 22904 (humphrey@cs.virginia.edu).* Dr. Humphrey received his Ph.D. degree in computer science from the University of Massachusetts in 1996. After spending two years as an assistant professor of computer science and engineering at the University of Colorado, he joined the University of Virginia in 1998. His research focuses on operating-system support for parallel, distributed, and real-time computation. He has created a real-time threads package that features novel semantics for hard real-time computation. He has also created operating-system support for distributed soft real-time computation, such as multimedia applications, addressing the ability to write, analyze, and execute applications that explicitly and dynamically adjust to fluctuating resource availability. Dr. Humphrey's current work is on providing operating-system or middleware support for large, heterogeneous virtual machines in the context of the Legion project, focusing on the general issues of computer security, resource management, and application design.

Anand Natrajan *Avaki Corporation, 15 New England Executive Park, Burlington, Massachusetts 01803 (anand@avaki.com).* Dr. Anand received his Ph.D. degree from the University of Virginia in 2000. He continued at the University of Virginia as a research scientist with the Legion project for another two years. Currently, he works as a Senior Software Engineer on grid systems for Avaki Corporation. His research focuses on harnessing the power of distributed systems for user applications. He has written tools for scheduling, running, and monitoring large numbers of legacy parameter-space jobs on a grid and for browsing the resources of a grid. Currently, Dr. Anand is designing and deploying a Web services interface for grids. He has published several related papers and participated in several fora related to grids. Prior to joining the Legion project, he worked on multi-representation modeling for distributed interactive simulation. His doctoral thesis addressed the problem of maintaining consistency among multiple concurrent representations of entities being modeled. In addition to distributed systems, he is interested in computer architecture and information retrieval.